

**REMARKS**

Reconsideration and allowance of the subject application are respectfully requested.

Applicant notes with appreciation the Examiner's withdrawal of the prior art rejections based on the previously-applied Bala reference. Applicants further appreciate the Examiner's acknowledgement of consideration of the information submitted with the Information Disclosure Statements filed on March 31, 2004 and June 3, 2004.

Claim 20 has been amended to substitute "said processing means" for "said processor core" to overcome the rejection under 35 U.S.C. §112, second paragraph.

Claims 1-5, 15-16, and 20 stand rejected under 35 U.S.C. §102(b) as being anticipated by newly-cited and applied Guccione. This rejection is respectfully traversed.

To establish that a claim is anticipated, the Examiner must point out where each and every limitation in the claim is found in a single prior art reference. *Scripps Clinic & Research Found. v. Genentec, Inc.*, 927 F.2d 1565 (Fed. Cir. 1991). Every limitation contained in the claims must be present in the reference, and if even one limitation is missing from the reference, then it does not anticipate the claim. *Kloster Speedsteel AB v. Crucible, Inc.*, 793 F.2d 1565 (Fed. Cir. 1986). Guccione fails to satisfy this exacting standard.

Java is a portable language and hardware independent. Thus, Java requires some mechanism for interfacing to either hardware or to non-Java code like device drivers and C or C++ libraries. Consider as an example a C programming language method incorporated into a Java program by defining an associated Java class function to be of type "native." The Java interpreter must provide an interface to enable execution of the native method.

Guccione describes three distinct native method interfaces including "stubs", Java native interface (JNI), and raw native interface (RNI). Because these three interfaces are incompatible,

Guccione provides C interface code to build three different interface libraries, known as dynamically-linked libraries (DLL), corresponding to each one of the three native interfaces, as illustrated in Figure 1.

In contrast, claims 1, 15, and 20 relate to an efficient way in which to execute a non-native portion of program code and to return either to a native calling program or a non-native calling program without the overhead of checking on each return whether the calling program was a native program or a non-native program.

The Examiner contends that the claim feature where an instruction translator is responsive to a return to a non-native instruction of the non-native instruction set to return processing to a non-native instruction is disclosed by Guccione on page 5 and Figure 5. Applicant disagrees. The program code of Figure 5 is C program code which specifies C function prototype for the native method "PrintNum." Using the Examiner's correlation of "native" to the C code and non-native to Java code, the return instruction in the program code of Figure 5 is an instruction in the native (C) programming language and **not** a return instruction in the non-native (Java) programming language.

There is another difference between claims 1, 15, and 20 and Guccione. The code segments of Figure 5 serve to return to execution of Java program code after having executed the C native method PrintNum. But this code segment does not serve as a special-purpose, return to non-native instruction of the non-native instruction set as specified in the independent claims, see e.g., integer (iii) in claim 1.

Nor does applicant accept the Examiner's assertion that Guccione implicitly discloses a non-native instruction set having a return to native instruction. As conceded by the Examiner, Guccione does not disclose the claim feature where the instruction translator is responsive to a

return to native instruction of the non-native instruction set to return processing to the native instruction, (see e.g., integer (iv) in claim 1). The Examiner asserts that since the function of the "Stubs Code" of Figure 5 of Guccione is to interface between non-native code and native code, it follows that a return to native instruction of the non-native instruction set must implicitly be disclosed by Guccione. We disagree. The Stubs code is intended to enable a native method to be incorporated in a non-native computer program. According, there is no need to provide a special-purpose, return to native instruction as specified, for example, by claim 1 integer (iv). Once Guccione's C method has executed, the program execution will invariably return to non-native code thereby obviating any need to provide a return to native instruction.

The Examiner equates the instruction translator of claim 1 with the dynamically-linked library (DLL) in Guccione. But a DLL is not an instruction translator. A DLL is simply a library linked to an application program when it is loaded or run. It is not an instruction translator operable to translate non-native instructions to native instructions.

Lacking multiple features from the independent claims 1, 15, and 20, the anticipation rejection should be withdrawn. Nor has the Examiner explained how the secondary references applied in the obviousness rejections overcome these fundamental deficiencies in Guccione. Guccione simply does not disclose or suggest an efficient way of returning from execution of a non-native subroutine to a calling program as described in claims 1, 15, and 20.

The application is now in condition for allowance. An early notice to that effect is earnestly solicited.

NEVILL

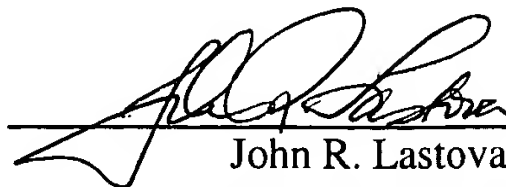
Appl. No. 09/887,561

November 24, 2004

Respectfully submitted,

**NIXON & VANDERHYE P.C.**

By:

A handwritten signature in black ink, appearing to read "John R. Lastova", is written over a horizontal line.

John R. Lastova

Reg. No. 33,149

JRL:at

1100 North Glebe Road, 8th Floor

Arlington, VA 22201-4714

Telephone: (703) 816-4000

Facsimile: (703) 816-4100